GYMNASIUM OTTOBRUNN

Oberstufenjahrgang 2017/19

Seminarfach Informatik

Seminararbeit

Kalender auf Basis von XML

Verfasser: Eric Brendel

Seminarleiter: OStR Peter Brichzin

Bewertung: …………. Punkte

Unterschrift des Seminarleiters: …………………………..

Inhaltsverzeichnis

[VORWORT 2](#_Toc528588269)

[EINLEITUNG 2](#_Toc528588270)

[XML UND DTD 2](#_Toc528588271)

[XML 3](#_Toc528588272)

[Was bedeutet das für die Implementierung? 3](#_Toc528588273)

[Welchen Nutzen kann der Programmierer daraus ziehen? 3](#_Toc528588274)

[DTD 3](#_Toc528588275)

[Notwendigkeit und Vorteile einer DTD 4](#_Toc528588276)

[Beispiel einer DTD (Auszug aus der CalML) 4](#_Toc528588277)

[TERMINKALENDER 6](#_Toc528588278)

[Produkt und Funktionen 6](#_Toc528588279)

[Beginn des Projekts 6](#_Toc528588280)

[User Stories 7](#_Toc528588281)

[Prototypen 8](#_Toc528588282)

[Zeitliche Planung 8](#_Toc528588283)

[Zusammenarbeit der Klassen (Klassendiagramm) 9](#_Toc528588284)

[DTD (eigene Sprache) 10](#_Toc528588285)

[Struktur 10](#_Toc528588286)

[Schnittstelle Java XML (Datenspeicherung) 11](#_Toc528588287)

[Wahl der verwendeten Bibliothek 11](#_Toc528588288)

[Implementierung 12](#_Toc528588289)

[Logic 13](#_Toc528588290)

[GUI mit JavaFX 13](#_Toc528588291)

[Entscheidung über Methode zur Erstellung der GUI 14](#_Toc528588292)

[Implementierung 14](#_Toc528588293)

[Fazit 16](#_Toc528588294)

[LITERATURVERZEICHNIS 16](#_Toc528588295)

[ANHÄNGE 17](#_Toc528588296)

[Installationshinweise 17](#_Toc528588297)

[Quelltext 17](#_Toc528588298)

# VORWORT

Diese Seminararbeit beschreibt das Entstehen und die Abläufe eines in Java geschriebenen Programms. Um das Geschriebene verstehen zu können, ist es daher notwendig, Grundkenntnisse in Java oder anderen objektorientierten Programmiersprachen zu haben. Vorkenntnisse zum Thema XML sind nicht notwendig, da der Erläuterung des Themas XML und DTD der Dokumentation vorangestellt ist.

# EINLEITUNG

In der heutigen Zeit wird auch innerhalb der Familie alles immer hektischer. Das eine Kind muss um 15.00 Uhr zum Fußball, das Andere zum Tennis und Mama hat sich abends zum Essen mit Freundinnen verabredet. Wenn solche Situationen der Alltag sind, dann kann es schon einmal zu Überschneidungen kommen. Kind A will sich mit Freunden treffen, dabei hat es aber schon einen Arzttermin, von dem es gar nichts weiß. Die Familie will einen Ausflug machen, doch das eine Kind hat aber am Tag darauf eine wichtige Matheklausur und noch gar nichts gelernt. So etwas kommt leider relativ häufig vor. Ein einfacher Kalender, der in der Küche oder im Wohnzimmer hängt, ist da nicht mehr der beste Weg. Es muss eine flexiblere Lösung her.

In dieser Seminararbeit wird der Entwicklungsprozess eines solchen digitalen Familienkalenders beschrieben. Dieser Kalender unterscheidet sich zudem durch einige praktische Funktionen, die der Familienfreundlichkeit dienen sollen, von einem herkömmlichen Kalender.

Die Erläuterung bezieht sich auf die Datenspeicherung und -verarbeitung, sowie die Visualisierung.

# XML UND DTD

Um einen guten Einstieg in die Thematik zu liefern und um das Verständnis für die folgende Seminararbeit zu verbessern, behandelt dieser kurze Abschnitt die Themen XML und DTD, die einen wichtigen Grundpfeiler für das mit der Arbeit zusammenhängende Projekt darstellen.

## 

## XML

Bei XML handelt es sich um die sogenannte „eXtensible Markup-Language“[[1]](#footnote-1). Bei einer Markupsprache handelt es sich nicht um eine Programmiersprache, wie beispielsweise Java, C++ oder Python, sondern um eine Auszeichnungssprache. Das „eXtensible“ ist Englisch und bedeutet „erweiterbar“. Es ist also möglich, die Sprache zu erweitern, bzw. eine neue Sprache zu erstellen und sie an die eigenen Ansprüche anzupassen.

### Was bedeutet das für die Implementierung?

Der wesentliche Unterschied zwischen einer Programmier- und einer Markup-/ Auszeichnungssprache besteht darin, dass es sich bei Letzterer um eine beschreibende Sprache handelt. Ein weiterer Unterschied besteht auch darin, dass mit Markupsprachen Dokumente erzeugt werden und mit Programmiersprachen Programme[[2]](#footnote-2). Für den Programmierer bedeutet das im Wesentlichen, dass eine Auszeichnungssprache die Struktur von Daten beschreibt und Programmiersprachen dem Realisieren logischer Abläufe dienen.

### Welchen Nutzen kann der Programmierer daraus ziehen?

XML ist ein sehr einfacher und praktischer Weg, Daten persistent zu speichern. Man kann mit Hilfe von Bibliotheken sogar ganze Objektzustände ohne größeren Aufwand speichern oder auslesen. Ein Vorteil ist hierbei auch, dass die Dateien nicht an eine Programmiersprache oder ein Betriebssystem gebunden, sondern von allen Betriebssystemen und Programmiersprachen benutzt werden können. Hierfür ist jedoch eine entsprechende Bibliothek notwendig. Als Beispiel für den Nutzen von XML dient die Sprache FXML. Mit Hilfe dieser, aus XML abgeleiteter, Sprache ist es möglich, Benutzeroberflächen für Java-Applikationen zu erstellen. Auch in diesem Projekt ist die Graphical User Interface(GUI) mit JavaFX und FXML erstellt.

## DTD

Bei einer DTD handelt es sich um eine Dokumenttyp-Definition (engl. Doctype-Definition). Mit Hilfe dieser können Regeln für das XML-Dokument definiert werden, um es zu gestalten. DTDs dienen also als eine Art Regelwerk für das XML-Dokument und definieren die darin vorhandenen Elemente und deren Attribute.[[3]](#footnote-4)

### Notwendigkeit und Vorteile einer DTD

Eine DTD ist nicht zwingender Weise für ein XML-Dokument notwendig. So ist das Einlesen und die Verarbeitung eines Dokuments problemlos ohne DTD möglich, dennoch bringt die Verwendung einige Vorteile mit sich:

* Ordnung / Struktur in dem Dokument: mit einer DTD erstellt man Regeln (für eine Struktur), die befolgt werden müssen, daher ist das Dokument übersichtlicher und strukturierter, was das Einlesen einfacher gestaltet.
* Keine Missverständnisse bei mehreren Programmierern: mit einer DTD kann sichergestellt werden, dass alle Programmierer bei dem Erstellen oder Verarbeiten des Dokuments der gleichen Struktur folgen. Damit sind Fehler und unterschiedliche Aufbauten der Dokumente ausgeschlossen.
* Mehrere Benutzer können die Referenzstruktur verwenden: Es gibt viele XML-Applikationen, die über eine DTD verfügen (z.B. FXML, CFD uvm.). Dank dieser können verschiedene Nutzer damit arbeiten, ohne jedes Mal eine neue Sprache erstellen zu müssen[[4]](#footnote-5).

Der Nutzen einer DTD ist also sichtlich groß. Allerdings sollte trotzdem überlegt werden, ob sie wirklich notwendig oder sinnvoll ist. Wenn das XML-Dokument weniger Elemente umfasst, als die DTD definiert, ist es sicher nicht sinnvoll, eine solche zu verwenden.

### Beispiel einer DTD (Auszug aus der CalML)

Bei CalML handelt es sich um die für das Projekt erstellte DTD. Sie dient der Speicherung der Ein Bild, das Elektronik, Computer, Laptop, drinnen enthält.

Mit hoher Zuverlässigkeit generierte BeschreibungKalenderdaten und legt die Regeln für das verwendete XML-Dokument zur Speicherung der Kalenderdaten fest. Bei dem rechts abgebildeten Quelltext sollte beachtet werden, dass es sich lediglich um einen Ausschnitt aus der gesamten DTD handelt, da das Beispiel sonst zu komplex wäre.

Mit der rechts abgebildeten DTD (Abb.1) ist das Speichern von Terminen verschiedener Personen möglich. Hier fallen die Schlüsselwörter !DOCTYPE, !ELEMENT und !ATTLIST sofort ins Auge:

Abbildung 1: Ausschnitt aus CalML

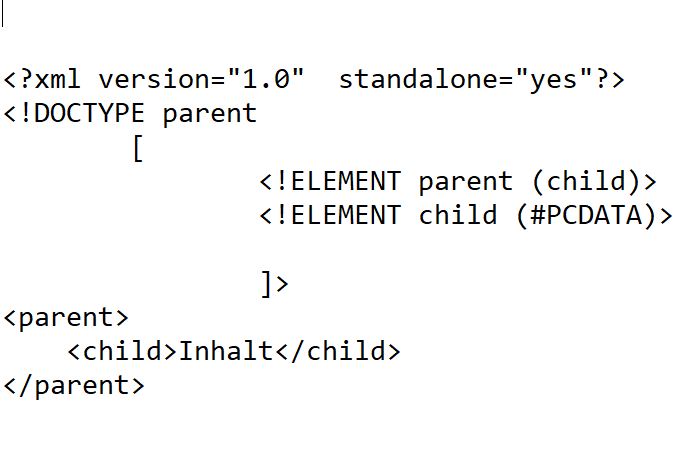
Mit dem Schlüsselwort !DOCTYPE wird ausgesagt, dass es sich bei mit den []-Klammern umschlossenen Bereich um die DTD handelt. Würde man das XML-Dokument übersetzen lassen, würde das Programm beim Einlesen der Datei diesen Teil nicht ausgeben. Ein Beispiel dafür liefern die beiden unterstehenden Abbildungen (Abb. 2 , Abb. 3)

Abbildung 2: Beispiel DTD

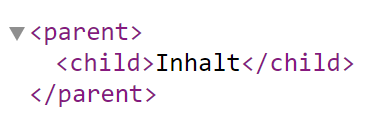


Abbildung 3: Ausgabe nach Einlesen

!ELEMENT dient der Definition eines Elements, das später im XML-Dokument benutzt werden kann. Der Bezeichner innerhalb der Klammer weist auf Unterelemente des entsprechenden Typs hin. In diesem Fall (Abb. 1) bedeutet das, dass in der Dokumenthierarchie nur day-Elemente unter einem month-Element stehen dürfen (Abb. 4). Der nachfolgende Operator legt die Quantität der Unterelemente fest: in diesem Beispiel liegt die Anzahl der Unterelemente zwischen null und theoretisch unendlich (gekennzeichnet durch den \*-Operator), wäre der physische Speicher nicht begrenzt.

Ein Bild, das Elektronik, Computer enthält.

Mit hoher Zuverlässigkeit generierte Beschreibung

Abbildung 4: Beispiel Ausschnitt XML-Dokument

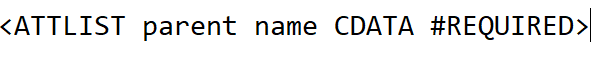
Mit !ATTLIST ist es möglich, ein Element mit Attributen zu versehen. Hierfür gibt man den Namen des Elements an, gefolgt von den Namen des Attributs, den Datentyp und einen Standardwert (Abb. 5). Auf den oben abgebildeten Fall bezogen (Abb. 1), ist der Datentyp CDATA, (ausgeschrieben Character Data), der mit einer Zeichenkette vergleichbar ist. Der Standardwert ist #REQURIED, was bedeutet, dass er immer angegeben werden muss und dieses Attribut nicht optional ist (dass er einen Pflichtwert/ Mandatory Value darstellt)

Abbildung 5: Beispiel ATTLIST

# TERMINKALENDER

Im folgenden Abschnitt sollen die Entstehung, die Funktionen und die dafür geschriebene Software erläutert werden. Eine kleine, kürzere Dokumentation ist in dem GitHub-Repository[[5]](#footnote-6) zu finden. Hier werden alle Funktionen und Attribute kurz und bündig erklärt, um das Verständnis beim Lesen des Codes zu erhöhen. Jedoch erfüllt die Seminararbeit den gleichen Zweck, nur ausführlicher.

## Produkt und Funktionen

### Beginn des Projekts

Die Planung ist bei einem Projekt dieser Größe maßgeblich. Neben der Planung der Programmierung und des Designs dürfen auf keinen Fall die Wünsche der Auftraggeber, Nutzer oder Kunden vernachlässigt werden.

Über die Aufgabenstellung beispielsweise wird eine persistente Datenspeicherung über XML verlangt. In Verbindung damit hat sich die verantwortliche Lehrkraft eine DTD gewünscht, damit das Projekt auf seiner eigenen Auszeichnungssprache basiert.

### User Stories

Da ein realer Auftraggeber mit Wünschen in diesem Sinne nicht vorhanden ist, sind die Funktionen aus Wünschen von Freunden und Verwandten entstanden. Bei dieser Umfrage sind folgende Ergebnisse hervorgegangen:

1. Selbstverständlich ist die individuelle Belegung und Anzeige der Termine der Grundpfeiler eines digitalen Kalenders. So passen sich die angezeigten Termine an den aktuellen Nutzer an. Auch bei dem Eintragen neuer Termine trägt das Programm diese nur bei dem aktuellen Nutzer ein.
2. Für die befragten Personen war es besonders wichtig, dass jeder seine Freizeit nutzten kann und diese somit fest in die Planung der Termine eingebunden ist. Die Möglichkeit der Buchung von wöchentlichen Freizeitphasen oder spezifische Zeiten als Freizeit festzulegen unterstützt dies. Möchte ein Nutzer einen, von einem anderen Nutzer belegten, Termin belegen, gibt das Programm eine Fehlermeldung aus, dass der Termin bereits belegt sei. Außerdem wird der Nutzer aufgefordert, die Person, die den Termin ursprünglich eingetragen hat, zu kontaktieren.
3. Eine weitere Funktion, die in den meisten Kalenderprogrammen Standard ist, sind Permanenttermine. So wird das Eintragen von sich täglich, wöchentlich, monatlich oder jährlich wiederholenden Terminen deutlich einfacher gemacht und damit die Nutzerfreundlichkeit gesteigert.
4. Eine besonders schülerfreundliche Funktion ist die Möglichkeit, den persönlichen Stundenplan eintragen zu können. Die Zeitabschnitte der Schulstunden, also zum Beispiel „08:00-08:45“, sind bereits definiert, um das Eintragen einfacher zu gestalten.
5. Ein weiterer wichtiger Teil des Kalenders ist die Möglichkeit, bereits erstellte Termine löschen zu können. Hierfür gibt der Nutzer Datum und Start des Termins ein und kann diesen dann löschen. Bei der Stundenplanfunktion ist das Löschen des Stundenplans eines oder mehrerer Tage direkt möglich.
6. Um die Familienfreundlichkeit zu erhöhen, gibt es die Möglichkeit Familientermine einzutragen. Diese überschreiben alle bereits vorhandenen Termine, das Überschreiben eines Familientermins ist jedoch nicht möglich. Familientermine werden immer für ganze Familie eingetragen.

### Prototypen

Ein Prototyp stellt in diesem Kontext einen programmbezogenen Meilenstein dar. Jeder Prototyp verfügt über neue Funktionen. Das Projekt verfügt über sechs implementierte Prototypen:

1. Der erste Prototyp bezieht sich nur auf die Schnittstelle XML. Hier ist lediglich das Ausgeben von einigen Informationen über die Konsole möglich, ohne dass diese in einer speziellen Form bearbeitet werden.
2. Der einzige Unterschied zwischen dem ersten und dem zweiten Prototyp liegt darin, dass die Rückgabe-Informationen nun über return erfasst werden,was einen wichtiger Schritt zur Einbindung in das gesamte Projekt darstellt .
3. Der dritte Prototyp liefert die ersten wichtigen Funktionen für den Kalender. Nun ist es möglich eigene Termine einzutragen. Es findet außerdem eine Überprüfung statt, ob die Zeitspanne entweder schon belegt oder der Termin nicht möglich ist.
4. Der vierte Prototyp ist der erste Schritt in Richtung Nutzerinteraktion. Die Benutzeroberfläche (GUI) ermöglicht dem Nutzer nun das Einsehen seiner Termine in geordneter Form und die Eingabe neuer Termine über ein eigenes Fenster.
5. Der fünfte Prototyp diente zur Verschönerung der GUI.
6. Der sechste Prototyp umfasst alle noch offenen UserStories, die oben aufgeführt sind. Auch Bugs, die im Laufe der Programmierung implementiert wurden, sollen gefixt werden. Auch das Ausführen der Datei ohne die Entwicklungsumgebung soll ermöglicht werden.

### Zeitliche Planung

Die zeitliche Planung der Seminararbeit fand im Rahmen der zweiten Zwischenaufgabe im W-Seminar statt. Hier war genau dies gewünscht. Die Verfasser sollten die von ihnen bereits aufgestellten Prototypen und Meilensteine mit Deadlines versehen. Das Ergebnis dieser Aufgabe war Folgendes[[6]](#footnote-7):

|  |  |
| --- | --- |
| 1. XML 100% verstehen: 01.04.2018 2. Programm starten: 01.04.2018 3. XML- Java Schnittstelle: 13.04.2018 4. Erster Prototyp: 15.04.2018 5. Zweiter Prototyp: 18.04.2018 6. Exposé: 25.04.2018 | 1. Gliederung: 25.04.2018 2. Dritter Prototyp: 02.05.2018 3. Erste GUI: 17.05.2018 4. Vierter Prototyp: 16.05.2018 5. Fünfter Prototyp: 23.05.2018 6. Sechster Prototyp: 15.06.2018 7. Probekapitel: 11.07.2018 |

Ein Bild, das Text, Karte enthält.

Mit sehr hoher Zuverlässigkeit generierte BeschreibungEs ist zu erkennen, dass die zeitliche Planung deutlich vor dem Abgabetermin der Seminararbeit endet. Die Zeitspanne bis zum Abgabetermin dient als „Puffer“. Für den Fall, dass mal eine Klausur oder andere wichtige Termine Priorität haben und daher nicht an der Seminararbeit gearbeitet werden kann, bleibt immer noch genug Zeit, um die noch ausstehende Arbeit nachzuholen. Dies soll dazu dienen, Stress zu vermeiden.

## Zusammenarbeit der Klassen (Klassendiagramm)

Die Implementierung der Klassen erfolgte nach der LDVCS-Architektur (Logic-Data-View-Controller-Service-Architektur), was bedeutet, dass die Klassen in fünf Themenbereiche aufgeteilt sind. „Ein Logikmodul reagiert auf Ereignisse, wie z. B. Benutzereingaben […]“[[7]](#footnote-8). Das Datenmodul dient zu persistenten Datenspeicherung. Im Falle dieses Projektes handelt es sich dabei um die XML-Datei, der zusätzlich die Klassen XMLReader, Writer und Reader zur Seite stehen, um die Daten zu schreiben und auszulesen. Über die View erfolgt die Darstellung, oder auch Präsentation, die in diesem Fall über die FXML-Dateien erfolgt. Zur Entgegennahme der Eingaben dient der/ die Controller, die in diesem Fall die Controllerklassen repräsentieren und die Eingabe über die GUI entgegennehmen, geringfügig filtern und anschließend an die Logic weitergeben. Der letzte, nicht implementierte Teil ist der Service, der zum Beispiel für die Arbeit mit Servern zuständig wäre.

Abbildung 6: vereinfachtes Klassendiagramm

Die Referenzstruktur der Klassen geht aus dem recht abgebildeten Klassendiagramm (Abb. 6) hervor. Eine ausführlichere Version ist auf dem beigelegten USB-Stick zu finden.

## DTD (eigene Sprache)

Die für das Projekt verwendete Sprache heißt CalML (Calendar Markup Language) und dient zu Speicherung der Termine und anderer Informationen, die für den Kalender notwendig sind.

### Struktur

Die DTD besteht im Wesentlichen aus drei Teilen:

1. Die Speicherung der Nutzer: Hier wird zum einen gespeichert, wer der aktuelle Nutzer ist, aber auch welche Nutzer generell in dem System gespeichert sein sollen. Dadurch wird ein schneller Zugriff auf die Namen der Nutzer ermöglicht, was besonders für die Datenverarbeitung sehr wichtig ist.
2. Die Permanenttermine: Hier werden Termine gespeichert, die sich täglich, wöchentlich, monatlich oder jährlich wiederholen. Der Grund für die getrennte Speicherung von den normalen/ einmaligen Terminen liegt darin, dass so die Speicherung deutlich einfacher und effizienter ist. Der Termin muss nur einmal abgespeichert werden und nicht je nach Wiederholung einmal pro Tag bis einmal pro Jahr. Außerdem ist so die erforderliche Rechenleistung beim Finden eines Termins deutlich geringer.
3. Speicherung der normalen /einmaligen Termine: Um das Suchen nach einem bestimmten Termin zu vereinfachen und eine geordnete Struktur in die DTD zu bringen, sind diese erst nach Jahr, anschließend nach Monat und zuletzt nach Tag unterteilt. Die Tagelemente können nun Perioden als Child-Elemente haben, in denen die Start- und Enduhrzeiten der Termine sowie Inhalte gespeichert werden. Um die Größe der Datei zu minimieren, werden Jahre, Monate und Tage nur dann erzeugt, wenn eine Periode in diesen angelegt werden soll. Existiert ein Tag-, Monat- oder Jahreselement bereits, so erstellt das Programm nicht extra einen neuen Node, sondern speichert den Termin in den bereits vorhandenen.

## Schnittstelle Java XML (Datenspeicherung)

Als Schnittstelle zwischen den Java-Klassen und dem XML-Dokument dient eine Bibliothek, die sich DOM nennt.

### Wahl der verwendeten Bibliothek

Zur Auswahl einer Schnittstelle Java-XML boten sich verschiedene Möglichkeiten: JAXB[[8]](#footnote-9), SAX und DOM[[9]](#footnote-10). Im Folgenden ist eine Tabelle zu sehen, in der die jeweiligen Vor- und Nachteile der verschiedenen Bibliotheken erläutert werden.

|  |  |  |
| --- | --- | --- |
|  | Vorteile | Nachteile |
| SAX | Performanter als DOM | Nicht standarisiert |
| Unflexibler Zugriff auf Elemente (nur erstes und letztes Element direkt auslesbar) |
| DOM | Flexibler Zugriff auf alle Elemente | Gesamtes XML-Dokument wird in den Hauptspeicher geladen |
| Direkter Zugriff über getElementById() oder getElementByTagName()auf Elemente |
| Modifikation des Dokuments sehr einfach |
| Standarisiert |
| JAXB | Ganze Objektzustände können gespeichert werden |  |
| Sehr performant |

Tabelle 1: Bibliotheken XML, Vergleich

Wie in der oberstehenden Tabelle (Tab. 1) zu erkennen ist, sticht JAXB als sehr positiv heraus. Die Geschwindigkeit der Datenverarbeitung von JAXB liegt weit über der von DOM, wie in der obenstehenden Tabelle zu erkennen ist, jedoch bietet DOM einige fallspezifische Vorteile. Zum einen ist es hier sehr einfach, Daten zu ändern und einzutragen, ohne ein gesamtes Objekt der Klasse zu erstellen und auch das Ändern einzelner Attribute funktioniert bei DOM deutlich einfacher. Außerdem ist die Anzahl der Nodes, die eingelesen oder geschrieben werden, von so geringer Größe, dass keine erkennbaren Verzögerungen auftreten und so die Benutzerfreundlichkeit und die Performance des Programmes nicht sichtbar beeinflusst werden. Dies ist auch der Grund, warum die Wahl letztendlich auf DOM und nicht auf JAXB fiel.

### Implementierung

Die Implementierung der Schnittstelle geschah über drei Klassen:

1. Die Klasse XMLReader stellt die direkte Schnittstelle zu der XML-Datei dar. Hier sind die notwendigen Imports für DOM eingebunden und hier werden alle Informationen direkt aus dem Dokument ausgelesen bzw. in das Dokument geschrieben. Die Funktionen beschränken sich hier auf das einfache Beschreiben und Lesen der Datei. Eine Überprüfung auf Vorhandensein des Nodes oder der überstehenden Nodes (Parents) findet nicht statt. Auch beim Beschreiben der Datei gibt es keine Prüfung dafür, ob ein solcher Node schon existiert.

Diese Aufgaben werden in den Klassen Reader und Writer übernommen.

1. Die Klasse Reader hat die Aufgabe, die in den Nodes gespeicherten Informationen in einen String zu übersetzen. Hierfür wird der entsprechende Node in ein Element umgewandelt. Mit Hilfe von Methoden der DOM-Bibliothek ist es nun möglich, die Informationen aus dem Node in Form eines Strings zu bekommen.
2. Die Klasse Writer übersetzt die String-Eingabe (des Nutzers über die GUI) in Nodes. Das bedeutet, dass die Funktionen dieser Klasse über die Eingabeparameter Attributwerte und Inhalte für die Elemente erhalten. Diese werden dann dem neuen Node hinzugefügt, der dann an den XMLReader weitergegeben wird. Der XMLReader schreibt diese anschließend in das XML-Dokument.

Die Schnittstelle zwischen Java und XML bzw. Logik (Klasse Logic) und XML ist nach diesem Konzept aufgebaut, da in der objektorientierten Programmierung jede Klasse ihre eigene Funktion haben soll. Dies bedeutet auch, dass jede Klasse nur eine Aufgabe hat. So werden die Klassen kleiner gehalten und der Quellcode der Funktionen übersichtlicher. Ein weiterer Vorteil ist, dass sich umfangreiche spezifische Funktionen aus den kleineren, vielfältig einsetzbaren Basisfunktionen aufbauen lassen.

## Logic

Die Logic-Klasse dient, wie der Name schon sagt, der Logik des Projektes. Sie verarbeitet die aus der GUI übergebenen Daten. Außerdem bündelt sie Informationen, um diese dann an die Benutzeroberfläche zu übergeben. Der Ursprung der Klasse liegt in der objektorientierten Programmierung. Da die Aufteilung der Funktionen dieser Klasse auf Reader und Writer einige Nachteile mit sich brächte, ist die Enscheidung gegen ein solchen Konzept gefallen. Hätten diese mehrere Aufgaben, würde das der objektorientierten Programmierung widersprechen. Ein Vorteil der gewählten Klassenstruktur ist auch, dass es mit dieser sehr einfach möglich ist, zu überprüfen, ob ein Termin bereits vorhanden ist, bevor es zum Eintragen kommt: Da es dabei notwendig ist, eine Referenz auf ein Reader-Objekt und ein Writer-Objekt zu haben, wäre diese Funktion ohne die Logic-Klasse nur schwer realisierbar.

## GUI mit JavaFX

Bei JavaFX handelt es sich um ein Framework zur Erstellung von Java-Applikationen. Hiermit ist das Erstellen von Benutzeroberflächen für verschiedene Plattformen[[10]](#footnote-11) möglich. JavaFX wurde entwickelt, um eine neue Alternative zu AWT und Swift[[11]](#footnote-12) zu liefern, die bis 2014 als Standardlösungen für grafische Anwendungen galten, jedoch einige Mängel besonders im Bereich Medien und Animationen aufwiesen. Seither dient JavaFX als Standard für die Entwicklung für GUIs.

### Entscheidung über Methode zur Erstellung der GUI

Es gibt verschiedene Möglichkeiten, eine GUI mit JavaFX zu erstellen.

1. Der wahrscheinlich einfachste Weg schien die Applikation SceneBuilder[[12]](#footnote-13) zu sein. Hier dient eine Benutzeroberfläche zum Erstellen der GUIs, die das Verschieben und Positionieren einzelner Elemente intuitiv ermöglich. Bei der Arbeit mit diesem Programm traten jedoch einige Fehler auf. Die Größenverhältnisse von Elementen und Fenstern wurden bei der Arbeit mit Nicht-Standardbildschirmauflösungen verzogen und nicht eingehalten. So waren die Elemente viel zu groß für das Fenster, obwohl die Vorschau im Programm selbst das gewünschte Ergebnis zeigte.
2. Als Alternative boten sich FXML-Dateien an. Diese haben zwei Vorteile: Zum einen erzeugt der SceneBuilder bereits solche Dateien, so dass nur noch das Anpassen der Größenverhältnisse erforderliche war. Zum anderen ist FXML aus XML abgeleitet und passt daher sehr gut zum Thema der Seminararbeit.
3. Eine weitere Möglichkeit bietet das Erstellen der Nutzeroberfläche über die Implementierung direkt in den Controllerklassen. Da diese aber so neben der Aufgabe der Weitergabe der Nutzereingabe an die Logic auch noch für das Anzeigen der GUI zuständig wären. Die spricht jedoch gegen das bereits oben eingeführte Logic-Data-View-Controller-Service-Paradigma, bei der die Controllerklassen nur zum Weiterleiten und leichten filtern der Eingabedaten dienen. An diesem Punkt ist jedoch zu erwähnen, dass die Implementierung der Farbdarstellung (blaue Labels und Knöpfe) genau über diesen Weg erfolgte, da eine Durchsetzung dieser über die FXML-Dateien recht kompliziert gewesen wäre.

### Implementierung

Die Einbindung der FXML-Dateien in das Projekt fand über sogenannte Controllerklassen statt[[13]](#footnote-14). Diese dienen dazu, die Daten, die über die Nutzereingabe erzeugt werden, zu erfassen, gegebenfalls zu filter, oder Ausgaben über die GUI festzulegen.

Die Imports import javafx.scene.\* und import javafx.fxml.\* binden die benötigten Bibliotheken in die Controllerklassen ein. Der erste Import ermöglicht das Ändern der aktuellen Szene. Hierzu lädt die Controllerklasse die entsprechende FXML-Datei ein und zeigt dann eine neue Scene als Fenster. Das aktuelle Fenster wird dabei geschlossen und auch temporär in den Controllerklassen gespeicherte Variablen werden hierbei verworfen, was jedoch kein Problem für die fallspezifische Programmierung darstellt. Der zweite Import dient der Kommunikation zwischen der FXML-Datei, die das Aussehen der GUI beschreibt, und ihrer Controllerklasse. Mit @FXML wird gekennzeichnet, dass das folgende Attribut/Objekt oder die folgende Methode in der FXML-Datei verankert ist. Die Controllerklasse definiert die Methodenrümpfe, die in der FXML-Datei deklariert werden und kann Methoden der Objekte aufrufen, die ebenfalls in der FXML-Datei deklariert werden. So kann zum Beispiel die Reaktion auf das Drücken eines Buttons in der Controllerklasse implementiert werden. Das unten abgebildete Klassendiagramm zeigt die Referenzen der Controllerklassen und FXML-Datei.

Da die Eingaben des Nutzers leider nicht immer so formuliert sind, dass das Programm sie verarbeiten kann, wird eine Fehlermeldung ausgegeben, wenn falsche Formulierungen oder Uneindeutigkeiten auftreten. Die Implementierung dieser Überprüfung steht jedoch in der Logic.

Ein Bild, das Text, Karte enthält.

Mit sehr hoher Zuverlässigkeit generierte Beschreibung

Abbildung 7: GUI, Klassendiagramm

## Fazit

Auch wenn es noch Verbesserungsmöglichkeiten gibt, ist das Projekt ein Erfolg. Die Implementierung umfasst alle UserStories und die Datenspeicherung umfasst verschiedene Möglichkeiten der Arbeit mit XML. Auch wenn die GUI optisch schlicht gehalten ist, erfüllt sie alle an sie gestellten Aufgaben. Als Verbesserung für die Überprüfung der Nutzereingabe wäre an eine formale Eingabesprache zu denken. So wäre es möglich, die Eingabe über einen entsprechenden Automaten zu überprüfen. Dies würde zum Einen die Aufteilung auf mehrere Klassen (objektorientierte Programmierung) verstärken und hätte zum Anderen eine logischere und geordnete Programmierung zur Folge, da diese bisher nur aus bedingten Anweisungen besteht, die aus dem Testen mit verschiedenen Eingaben hervorgegangen sind. Das Ziel, einen digitalen Familienkalender zu programmieren, wurde ebenfalls erreicht, da es einige Funktionen gibt, die insbesondere für Familien hilfreich sein können, wie zum Beispiel die Möglichkeit, eine Begleitperson einzutragen, oder Familienevents einzutragen. Dies beweist auch eine 2-wöchige Testphase in zwei Familien, aus denen ein positives Feedback hervorgin.

# LITERATURVERZEICHNIS

* Boumphrey, F., Tittel, E., XML für Dummies, New York, mitp, 2002
* Brendel, E., W-Seminar: Familienkalender (Eric Brendel),  
  <https://trello.com/b/Hg1mSbSR/w-seminar-familienkallender-eric-brendel> [

Stand 25.10.2018

Abrufdatum 25.10.2018

* Brendel, E., Reiswaffl/Kalender,   
  <https://github.com/Reiswaffl/Kalender>

Stand 25.10.2018

Abrufdatum 25.10.2018

* Hauser, T., XML-Standards, Paderborn, entwickler.press, 2010
* Hochschule Augsburg, Logic-Data-View-Controller-Service-Paradigma,

<https://glossar.hs-augsburg.de/Logic-Data-View-Controller-Service-Paradigma>

Stand 22.09.2017 16:49 Uhr

Abrufdatum 25.10.2018

* Schindler, J., JavaFx Java GUI Tutorial[0]-[4], [www.youtube.com/watch?v=fWwaWCkh9Ig&list=PLNIWFrcmR15Jdevz2v30YipAJq0hnh2xM](http://www.youtube.com/watch?v=fWwaWCkh9Ig&list=PLNIWFrcmR15Jdevz2v30YipAJq0hnh2xM)

Stand 07.07.2015

Abrufdatum 25.10.2018

* o.A, JavaFX SceneBuilder – A Visual Layout Tool for JavaFX Applikations,

<https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html>

Stand o.D.

Abrufdatum 25.10.2018

* Oracle und/oder Partner, When to Use Dom,  
  <https://docs.oracle.com/javase/tutorial/jaxp/dom/when.html>

Stand 2017

Abrufdatum 25.20.2018

# ANHÄNGE

Einige Anhänge, wie zum Beispiel der gesamte Quellcode befinden sich auf dem beigelegten USB-Stick. Hier sind auch die Klassendiagramme, sowie das ausführbare Programm zu finden. Eine Dokumentation des Quelltextes ist auf dem zu den GitHub gehörenden Wiki zu finden, obwohl der Quelltext auch kommentiert ist.

## Installationshinweise

## Quelltext

1. vgl. „XML Standards“ [Ha10] [↑](#footnote-ref-1)
2. <http://webkompetenz.wikidot.com/wp:auszeichnungs-programmiersprachen>, [Mü09] [↑](#footnote-ref-2)
3. „XML für Dummies“, [vgl. TB02] [↑](#footnote-ref-4)
4. „XML für Dummies“, [vgl. TB02] [↑](#footnote-ref-5)
5. GitHub-Wiki: <https://github.com/Reiswaffl/Kalendar/wiki> [Br19] [↑](#footnote-ref-6)
6. Vgl. Trelloboard zur zeitlichen Planung: <https://trello.com/b/Hg1mSbSR/w-seminar-familienkallender-eric-brendel> [Br19] [↑](#footnote-ref-7)
7. https://glossar.hs-augsburg.de/Logic-Data-View-Controller-Service-Paradigma [↑](#footnote-ref-8)
8. <https://docs.oracle.com/javase/tutorial/jaxp/dom/when.html> [↑](#footnote-ref-9)
9. SAX und DOM in Tabelle aus „XML Standards“, [vgl. Ha10] [↑](#footnote-ref-10)
10. Windows, macOs und Linux [↑](#footnote-ref-11)
11. //todo [↑](#footnote-ref-12)
12. <https://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html> [↑](#footnote-ref-13)
13. vgl.

    [www.youtube.com/watch?v=fWwaWCkh9Ig&list=PLNIWFrcmR15Jdevz2v30YipAJq0hnh2xM](http://www.youtube.com/watch?v=fWwaWCkh9Ig&list=PLNIWFrcmR15Jdevz2v30YipAJq0hnh2xM) [Sh15] [↑](#footnote-ref-14)